Matlab-Beispiel-Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cascadeIIR function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function cascadeIIR

% --- set default parameters ---
coef_size = 16;
fd = 150 ;

% ----------------------------------------------------------
% --- high pass filter design with single 6th order IIR ---
% ----------------------------------------------------------

f_hpf = [0  0,5 0,5 1];
m_hpf = [0  0   1   1] ;

[b_hpf, a_hpf] = yulewalk(6,f_hpf,m_hpf);

% Find the roots of the polynomials
p_hpf=roots(a_hpf);
z_hpf=roots(b_hpf);

%Quantization
b_hpf = quant(b_hpf,coef_size);
a_hpf = quant(a_hpf,coef_size);

% Plot the overall desired Frequency response
figure(1);
hold off;
freq_resp(b_hpf,a_hpf,fd,'r');
title(‚Desired high pass filter with single 6th order IIR');

%figure(2);
%hold on;
%plot(a_hpf, ‚r');
%plot(b_hpf, ‚b');
%title(‚Single 6th Pole coefficients');

fprintf(1,'Single 6th order high pass filter ax= ‚)
hex(a_hpf);
fprintf(1,'\n');
fprintf(1,'Single 6th order high pass filter bx = ‚)
hex(b_hpf);
fprintf(1,'\n');

% -------------------------------------------------------------
% --- Implement 2nd order 1st stage and 4th order 2nd stage     ---
% -------------------------------------------------------------

% Two poles + Two zeros for stability
a1_hpf(1)=1;
a1_hpf(2)=-2*real(p_hpf(1));
a1_hpf(3)=real(p_hpf(1)).^2 + imag(p_hpf(1)).^2;

b1_hpf(1)= b_hpf(1);
b1_hpf(2)= -2*real(z_hpf(1))*b_hpf(1);
b1_hpf(3)= b_hpf(1)*((real(z_hpf(1))).^2 + (imag(z_hpf(1))).^2)

% Plot the 2nd order filter
```

```
figure(3);
hold off;
freq_resp(b1_hpf,a1_hpf,fd,'r');
title(,1st stage pass filter with single 2th order IIR');

% Create the remaining 4 pole + 4 zero filter
a1_temp1(1)=1;
a1_temp1(2)=-2*real(p_hpf(3));
a1_temp1(3)=real(p_hpf(3)).^2 + imag(p_hpf(3)).^2;
b1_temp1(1)= 1;
b1_temp1(2)= -2*real(z_hpf(3));
b1_temp1(3)= ((real(z_hpf(3))).^2 + (imag(z_hpf(3))).^2)

a1_temp2(1)=1;
a1_temp2(2)=-2*real(p_hpf(5));
a1_temp2(3)=real(p_hpf(5)).^2 + imag(p_hpf(5)).^2;
b1_temp2(1)= 1;
b1_temp2(2)= -2*real(z_hpf(5));
b1_temp2(3)= ((real(z_hpf(5))).^2 + (imag(z_hpf(5))).^2)

a2_hpf=conv(a1_temp1,a1_temp2);
b2_hpf=conv(b1_temp1,b1_temp2);

% Plot the 4th filter
figure(4);
hold off;
freq_resp(b2_hpf,a2_hpf,fd,'r');
title(,2nd stage high pass filter with single 4th order IIR');

fprintf(1,'2th order high pass filter ax= ,)
hex(a1_hpf);
fprintf(1,'\n');
fprintf(1,'2th order high pass filter bx = ,)
hex(b1_hpf);
fprintf(1,'\n');

fprintf(1,'4th order high pass filter ax= ,)
hex(a2_hpf);
fprintf(1,'\n');
fprintf(1,'4th order high pass filter bx = ,)
hex(b2_hpf);
fprintf(1,'\n');

H1=dfilt.df1(b1_hpf, a1_hpf);
H2=dfilt.df1(b2_hpf, a2_hpf);
HCOMB1=dfilt.cascade(H1,H2);
fvtool(HCOMB1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% quant function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = quant(x,bits)

% Quantize vector to the number of bits specified
% using round funciton.
%
% x     - vector to be quantized (-1,0 to .9999)
% bits  - bits of quantization
% y     - quantized output (-1,0 to .9999)

% --- calculate number of quantization levels ---
```

```
levels = 2^(bits-1);
max = levels-1;
min = -levels;

% --- convert to integer ---

y=round(x*levels);

% --- limit signal ---

for i = 1:length(y)
  if(y(i)>max)
    y(i)=max;
  elseif(y(i)<min)
    y(i)=min;
  end
end

% --- convert back to proper range ---

y = y/levels;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% freq_resp function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [h_db,f] = freq_resp(b,a,fs,plot_opt)

% [h_db,f] = freq_resp(b,a,fs,plot_opt)
%
% Calculates and plots the response of a filter.

if(nargin < 4)
   plot_opt = ,b';
end

if(nargin < 3)
   fs = 8000;
end

[h,f] = freqz(b,a,65536,fs);
h_db = 20*log10(abs(h));
plot(f,h_db,plot_opt);
axis([0 fs/2 (max(h_db)-140) max(h_db)+5]);
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% hex function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hex(x,int_flag)

% hex(x,int_flag)
%
% Display the vector x in hexadecimal.
% If int_flag = 0 (default) then x is assumed to be
% fractional, otherwise  it is assumed to be 16 bit integer.

if(nargin < 2)
  int_flag = 0;
end

if(int_flag == 0)
   x = frac2int(x);
```

```
    else
       x = round(x);
    end

    fprintf(1,'\n');
    fprintf(1,'0x%04x\n',x);
    fprintf(1,'\n');


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % frac2int function
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function y = frac2int(x,bits)

    % y = frac2int(x,bits)
    %
    % Converts a real valued vector to its equivalent fractional
    % integer value. Values outside the range -1 to 1 are clipped.
    % The number of quantization bits defaults to 16.

    if(nargin < 2)
      bits = 16;
    end

    x_ = quant(x,bits);
    half_range = 2^(bits-1);
    full_range = 2^bits;

    y = zeros(length(x),1);

    for i = 1:length(x)
      if(x_(i)<0)
        y(i) = x_(i)*half_range + full_range;
      else
        y(i) = x_(i)*half_range;
      end
end
```